50 algorithmic exercises in JAVA

50 Practical Exercises
to Develop Your Programming
Programming Skills

50 ALGORITHMIC EXERCISES IN JAVA

INTRODUCTION

Algorithms are at the heart of computer science. It is the art of solving problems using sequences of logical instructions. Whether you are a beginner looking to master the basics of programming or an experienced developer looking to improve your problem-solving skills, this book is designed for you.

STATEMENT: BUBBLE SORTING IN JAVA

Write a Java program to sort an array of integers using the bubble sort algorithm. The bubble sort algorithm works by comparing and swapping adjacent elements if they are in the wrong order, and repeats this process until the whole array is sorted.

You need to code the bubble sort logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class TriBubbles {

   public static void main(String[] args) {

       Scanner scanner = new Scanner(System.in);

       System.out.print("Enter the size of the table: ");

       int n = scanner.nextInt();

       int[] array = new int[n];

       System.out.println("Enter the elements of the table:");

       for (int i = 0; i < n; i++) {

           array[i] = scanner.nextInt();

       }

       // Implementation of the bubble sorting algorithm
```

```
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
        System.out.println("Table sorted:");
        for (int i = 0; i < n; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

In this example, we have used an array of integers to store the elements, and the bubble sort algorithm is implemented inside two for loops. The program asks the user to enter the size of the array and the elements of the array, then displays the sorted array at the end.

STATEMENT:
QUICKSORT IN JAVA

Write a Java program to sort an array of integers using the quicksort algorithm. The quicksort algorithm is an efficient sorting algorithm based on the divide-and-conquer method. It selects a pivot element, partitions the array into two sub-arrays (elements below the pivot and elements above the pivot), then recursively sorts these sub-arrays.

You need to code the quick sort logic in Java and make sure your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class QuickSort {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the size of the table: ");

      int n = scanner.nextInt();

      int[] array = new int[n];

      System.out.println("Enter the elements of the table:");

      for (int i = 0; i < n; i++) {

         array[i] = scanner.nextInt();

      }
```

```
        // Call up the quick sort function
        quickSort(array, 0, n - 1);
        System.out.println("Table sorted:");
        for (int i = 0; i < n; i++) {
            System.out.print(array[i] + " ");
        }
    }
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);
            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }
    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
```

```
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }
}
```

In this example, we have used the quicksort method to sort an array of integers. The program asks the user to enter the size of the array and the elements, then uses recursion to sort the array.

STATEMENT:
MERGESORT IN JAVA

Write a Java program to sort an array of integers using the merge sort algorithm (Mergesort). The merge sort algorithm is based on the "divide and conquer" method by dividing the array into two halves, sorting these halves, then merging the sorted halves to obtain a sorted array.

You need to code the fusion sorting logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
public class MergeSort {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the size of the table: ");
      int n = scanner.nextInt();
      int[] array = new int[n];
      System.out.println("Enter the elements of the table:");
      for (int i = 0; i < n; i++) {
         array[i] = scanner.nextInt();
      }
```

```
        // Call the merge sort function
        mergeSort(array, 0, n - 1);
        System.out.println("Table sorted:");
        for (int i = 0; i < n; i++) {
            System.out.print(array[i] + " ");
        }
    }
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int middle = (left + right) / 2;
            mergeSort(arr, left, middle);
            mergeSort(arr, middle + 1, right);
            merge(arr, left, middle, right);
        }
    }
    public static void merge(int[] arr, int left, int middle, int right) {
        int n1 = middle - left + 1;
        int n2 = right - middle;
        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];
        for (int i = 0; i < n1; i++) {
            leftArray[i] = arr[left + i];
        }
```

```
        for (int j = 0; j < n2; j++) {
            rightArray[j] = arr[middle + 1 + j];
        }
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                arr[k] = leftArray[i];
                i++;
            } else {
                arr[k] = rightArray[j];
                j++;
            }
            k++;
        }
        while (i < n1) {
            arr[k] = leftArray[i];
            i++;
            k++;
        }
        while (j < n2) {
            arr[k] = rightArray[j];
            j++;
            k++;
```

```
        }
    }
}
```

In this example, we have used the Mergesort algorithm to sort an array of integers. The program asks the user to enter the size of the array and the elements, then uses recursion to sort the array by dividing, sorting and merging the sub-arrays.

STATEMENT: SORTING BY SELECTION IN JAVA

Write a Java program to sort an array of integers using the selection sort algorithm. The selection sort algorithm consists of finding the smallest element in the array and placing it in the first position, then repeating this process for the rest of the array.

You need to code the sort-by-selection logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class SelectionSort {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the size of the table: ");

      int n = scanner.nextInt();

      int[] array = new int[n];

      System.out.println("Enter the elements of the table:");

      for (int i = 0; i < n; i++) {

         array[i] = scanner.nextInt();

      }

      // Call up the sort by selection function
```

```
        selectionSort(array);
        System.out.println("Table sorted:");
        for (int i = 0; i < n; i++) {
            System.out.print(array[i] + " ");
        }
    }
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}
```

In this example, we have used the selection sort algorithm to sort an array of integers. The program asks the user to enter the size of the array and the elements, then uses a double loop to find the minimum element in the

unsorted portion of the array and places it in the correct position. This operation is repeated until the entire array is sorted.

STATEMENT: INSERTION SORTING IN JAVA

Write a Java program to sort an array of integers using the insertion sort algorithm. The insertion sort algorithm consists of successively inserting each unsorted element in its correct position among the sorted elements, from left to right.

You need to code the insertion sort logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class InsertionSort {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the size of the table: ");

      int n = scanner.nextInt();

      int[] array = new int[n];

      System.out.println("Enter the elements of the table:");

      for (int i = 0; i < n; i++) {

         array[i] = scanner.nextInt();

      }

      // Call the insertion sort function

      insertSort(array);
```

```
        System.out.println("Table sorted:");
        for (int i = 0; i < n; i++) {
            System.out.print(array[i] + " ");
        }
    }
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
}
```

In this example, we have used the insertion sort algorithm to sort an array of integers. The program asks the user to enter the size of the array and the elements, then uses a loop to insert each unsorted element in its correct position among the sorted elements. This operation is repeated until the entire array is sorted.

STATEMENT: BINARY SEARCH IN JAVA

Write a Java program that performs a binary search in a sorted array of integers. Binary search is an efficient algorithm that searches for an element in a sorted array by recursively dividing the array into two parts and comparing the element searched for with the element in the middle. If the element searched for is equal to the element in the middle, the algorithm returns the position of this element in the array. Otherwise, the algorithm reduces the search to the half of the array in which the element could be found.

You need to code the binary search logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class BinarySearch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the sorted array: ");

        int n = scanner.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the sorted array (in ascending order):");

        for (int i = 0; i < n; i++) {
```

```
            array[i] = scanner.nextInt();
        }
        System.out.print("Enter the item to search for: ");
        int elementSearch = scanner.nextInt();
        int position = binarySearch(array, elementSearch);
        if (position != -1) {
            System.out.println("The element " + elementSearch + " has been
found at position " + position + " in the array.");
        } else {
            System.out.println("The element " + elementSearch + " was not found
in the array.");
        }
    }
    public static int binarySearch(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target) {
                return mid;
            }
            if (arr[mid] < target) {
                left = mid + 1;
            } else {
```

```
            right = mid - 1;
        }
    }
    return -1;
  }
}
```

In this example, we have used the binary search algorithm to find an element in a sorted array. The program asks the user to enter the size of the sorted array, the elements of the array and the element to be searched for. It then performs the binary search to find the element and displays its position if it is found, or indicates if it has not been found.

STATEMENT: SEQUENTIAL SEARCH IN JAVA

Write a Java program to perform a sequential search in an array of integers. Sequential search is a simple algorithm that consists of going through the array element by element until the element you are looking for is found or the entire array has been searched.

You need to code the sequential search logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class SequentialSearch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the table: ");
        int n = scanner.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the table:");
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
        }
```

```
        System.out.print("Enter the item to search for: ");
        int elementSearch = scanner.nextInt();
        int position = sequentialSearch(array, elementSearch);
        if (position != -1) {
            System.out.println("The element " + elementSearch + " has been
found at position " + position + " in the array.");
        } else {
            System.out.println("The element " + elementSearch + " was not found
in the array.");
        }
    }
    public static int sequentialSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i; // Element found, returns position
            }
        }
        return -1; // Element not found
    }
}
```

In this example, we have used the sequential search algorithm to find an element in an array. The program asks the user to enter the size of the array, the elements of the array and the element to be searched for. It then performs the sequential search to find the element and displays its position if it is found, or indicates if it has not been found. Sequential search is simple but

can be less efficient than other methods for large arrays, as it traverses all the elements in the array.

STATEMENT: CALCULATING THE FACTORIAL OF A NUMBER IN JAVA

Write a Java program that calculates the factorial of a given number. The factorial of a positive integer n is the product of all the positive integers from 1 to n.

You need to code the logic for calculating the factorial in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class Factorial {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter a positive integer: ");

      int n = scanner.nextInt();

      if (n < 0) {

         System.out.println("The factorial is only defined for positive integers.");
```

```
        } else {
            long result = calculateFactorial(n);
            System.out.println("The factorial of " + n + " is " + result);
        }
    }
    public static long calculateFactorial(int n) {
        if (n == 0 || n == 1) {
            return 1; // The factorial of 0 and 1 is 1
          }
        long factorial = 1;
        for (int i = 2; i <= n; i++) {
            factorial *= i;
        }
        return factorial;
    }
}
```

In this example, we have used a for loop to calculate the factorial of a given positive integer. The program asks the user to enter a number, then checks whether it is positive. If it is, it calculates the factorial by multiplying the integers from 1 to n, otherwise it indicates that the factorial is only defined for positive integers. The factorial of 0 and 1 is defined as being equal to 1.

STATEMENT: CALCULATING THE SUM OF A SERIES OF INTEGERS IN JAVA

Write a Java program that calculates the sum of a series of integers. The user must enter the number of integers to be added together, then enter the integers. The program must then calculate the sum of these integers and display it.

You need to code the logic for calculating the sum in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class SumOfIntegers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of integers to be added: ");
        int n = scanner.nextInt();
        int[] integers = new int[n];
        int sum = 0;
        System.out.println("Enter the integers one by one:");
```

```
        for (int i = 0; i < n; i++) {
            integers[i] = scanner.nextInt();
            sum += integers[i];
        }
        System.out.println("The sum of the integers is: " + sum);
    }
}
```

In this example, we have used a for loop to allow the user to enter the integers one at a time. With each iteration of the loop, the integer entered is added to the sum variable, which is used to store the total sum. Once all the integers have been entered, the program displays the sum.

STATEMENT: CALCULATING THE POWER OF A NUMBER IN JAVA

Write a Java program that calculates the power of a number. The user must enter the base (a number) and the exponent (a non-negative integer). The program must calculate the value of the base raised to the power of the exponent and display it.

You need to code the logic for calculating the power in Java, also handling the case where the exponent is a negative number, and make sure that your program works correctly. You can add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class PowerCalculation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the base (a number): ");
        double base = scanner.nextDouble();
        System.out.print("Enter the exponent (a non-negative integer): ");
        int exponent = scanner.nextInt();
        if (exponent < 0) {
```

```
            System.out.println("The exponent must be a non-negative integer.");
        } else {
            double result = calculatePower(base, exponent);
            System.out.println(base + "raised to the power" + exponent +
"equals" + result);
        }
    }
    public static double calculatePower(double base, int exponent) {
        if (exponent == 0) {
            return 1;
        } else {
            double result = 1;
            for (int i = 1; i <= exponent; i++) {
                result *= base;
            }
            return result;
        }
    }
}
```

In this example, we have used a for loop to calculate the power of the base raised to the exponent. The program asks the user to enter the base and exponent, checks that the exponent is a non-negative integer, then calculates the power and displays it. The power of the base to the 0th power is defined as 1.

STATEMENT: CALCULATING THE GREATEST COMMON DIVISOR (GCD) OF TWO NUMBERS IN JAVA

Write a Java program that calculates the greatest common divisor (GCD) of two given numbers. The GCD is the largest integer that divides both numbers without leaving a remainder.

You need to code the logic for calculating the PGCD in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class GCDCalculation {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the first number: ");

      int number1 = scanner.nextInt();

      System.out.print("Enter the second number: ");

      int number2 = scanner.nextInt();
```

```
        int gcd = calculateGCD(number1, number2);
        System.out.println("The PGCD of " + number1 + " and " + number2 + " is " + gcd);
    }
    public static int calculateGCD(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
}
```

In this example, we have used Euclid's algorithm to calculate the GCD of two numbers. The program asks the user to enter the two numbers, then uses a while loop to repeat the steps of Euclid's algorithm until b becomes zero. The PGCD is then equal to a. The programme then displays the PGCD of the two numbers entered.

STATEMENT: CALCULATION OF THE LEAST COMMON MULTIPLE (LCMM) OF TWO NUMBERS IN JAVA

Write a Java program that calculates the least common multiple (LCMM) of two given numbers. The LCMM is the smallest positive integer that is a common multiple of the two numbers.

You need to code the PPCM calculation logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class LCMCalculation {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the first number: ");

        int number1 = scanner.nextInt();

        System.out.print("Enter the second number: ");

        int number2 = scanner.nextInt();

        int lcm = calculateLCM(number1, number2);
```

```
        System.out.println("The PPCM of " + number1 + " and " + number2 + " is " + lcm);
    }
    public static int calculateLCM(int a, int b) {
        int gcd = calculateGCD(a, b);
        return (a * b) / gcd;
    }
    public static int calculateGCD(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
}
```

In this example, we have used Euclid's algorithm to calculate the PGCD of two numbers, then we have used the relationship between the PGCD and the PPCM to calculate the PPCM. The program asks the user to enter the two numbers, then calculates the PPCM using the formula (a * b) / gcd(a, b). The PPCM is then displayed.

STATEMENT : SEARCH FOR PRIME NUMBERS IN A GIVEN RANGE IN JAVA

Write a Java program that finds and displays all the prime numbers in a given range. A prime number is an integer greater than 1 that has only two divisors: 1 and itself.

You need to code the logic for finding prime numbers in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class PrimeNumbersInRange {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the minimum value for the range: ");
        int min = scanner.nextInt();
        System.out.print("Enter the maximum value for the range: ");
        int max = scanner.nextInt();
        if (min < 2) {
            min = 2; // 2 is the first prime number
        }
```

```
        System.out.println("Prime numbers in the range " + min + " to " + max +
" :");
        findAndDisplayPrimes(min, max);
    }
    public static void findAndDisplayPrimes(int min, int max) {
        for (int number = min; number <= max; number++) {
            if (isPrime(number)) {
                System.out.print(number + " ");
            }
        }
    }
    public static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

In this example, we have used an isPrime function to check whether a given number is prime. The program asks the user to enter the minimum and maximum value in the range, and then finds and displays all the primes in that range using the findAndDisplayPrimes function. The isPrime function checks whether a number is prime by checking whether it is divisible by a number other than 1 and itself. Note that 2 is considered the first prime number, and the program ensures that the minimum value in the range is at least 2.

STATEMENT: CALCULATING THE SUM OF THE DIGITS OF A NUMBER IN JAVA

Write a Java program that calculates the sum of the digits of an integer. The user enters an integer, and the program calculates the sum of its digits.

You need to code the logic for calculating the sum of the numbers in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class SumOfDigits {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();
        int sum = calculateSumOfDigits(number);
        System.out.println("The sum of the numbers in " + number + " is " + sum);
    }
    public static int calculateSumOfDigits(int num) {
```

```
        int sum = 0;
        while (num != 0) {
            int digit = num % 10; // Get the last digit
            sum += digit; // Add the digit to the sum
            num /= 10; // Delete the last digit
        }
        return sum;
    }
}
```

In this example, we have used a while loop to calculate the sum of the digits of an integer. The program asks the user to enter a number, then uses the while loop to extract the last digit at each iteration, add it to the sum and remove that digit from the number. The loop repeats until the number becomes zero. The program then displays the sum of the digits.

STATEMENT: CALCULATING THE NUMBER OF DIGITS IN AN INTEGER IN JAVA

Write a Java program that calculates the number of digits in an integer. The user enters an integer, and the program counts and displays the number of digits in the integer.

You need to code the logic for calculating the number of digits in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class CountDigits {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter an integer: ");

      int number = scanner.nextInt();

      int count = countDigits(number);

      System.out.println("The number " + number + " contains " + count + " digits.");

   }
```

```
    public static int countDigits(int num) {
        int count = 0;
        while (num != 0) {
            count++; // Increment the counter at each iteration
            num /= 10; // Delete the last digit
        }
        return count;
    }
}
```

In this example, we have used a while loop to count the number of digits in an integer. The program asks the user to enter a number, then uses the while loop to increment the counter on each iteration and delete the last digit of the number. The loop repeats until the number becomes zero. The program then displays the number of digits in the number entered.

STATEMENT : FINDING THE MINIMUM OR MAXIMUM IN AN ARRAY IN JAVA

Write a Java program that finds and displays either the minimum or the maximum of an array of integers. The user must first choose whether to search for the minimum or the maximum, and then enter the elements of the array. The program must then find and display the corresponding value (minimum or maximum) in the array.

You need to code the minimum or maximum search logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class MinMaxInArray {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Do you want to search for the minimum or maximum? (min/max) : ");

      String choice = scanner.nextLine();

      if (choice.equalsIgnoreCase("min")) {

         findAndDisplayMinimum();
```

```
    } else if (choice.equalsIgnoreCase("max")) {
        findAndDisplayMaximum();
    } else {
        System.out.println("Invalid choice. Please enter 'min' or 'max'.");
    }
}
public static void findAndDisplayMinimum() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the size of the table: ");
    int n = scanner.nextInt();
    int[] arr = new int[n];
    System.out.println("Enter the elements of the table:");
    for (int i = 0; i < n; i++) {
        arr[i] = scanner.nextInt();
    }
    int min = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
    }
    System.out.println("The minimum in the table is: " + min);
}
```

```
    public static void findAndDisplayMaximum() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the table: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the table:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        int max = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        System.out.println("The maximum of the array is: " + max);
    }
}
```

In this example, we have created a program that allows the user to choose whether to search for the minimum or the maximum in an array of integers. The program first asks for this choice, followed by the size of the array and the elements of the array. Depending on the user's choice, the program searches for and displays either the minimum or the maximum. The code is divided into two functions, findAndDisplayMinimum and findAndDisplayMaximum, to handle the two cases.

STATEMENT: INVERTING A CHARACTER STRING IN JAVA

Write a Java program that inverts a string of characters. The user enters a string, and the program inverts the characters in the string and displays the result.

You need to code the string inversion logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
public class ReverseString {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a character string: ");
        String input = scanner.nextLine();
        String reversed = reverseString(input);
        System.out.println("The reversed string is: " + reversed);
    }
    public static String reverseString(String input) {
        char[] charArray = input.toCharArray();
```

```
        int left = 0;
        int right = input.length() - 1;
        while (left < right) {
            char temp = charArray[left];
            charArray[left] = charArray[right];
            charArray[right] = temp;
            left++;
            right--;
        }
        return new String(charArray);
    }
}
```

In this example, we have used a character array to store the characters in the string and then used a two pointer technique to invert the characters. The program asks the user to enter a string, then uses a while loop to invert the characters by swapping the characters in the left and right positions. The inverted string is then displayed.

STATEMENT: PALINDROME VERIFICATION IN JAVA

Write a Java program that checks whether a character string is a palindrome. A palindrome is a sequence of characters that reads the same from left to right and right to left, ignoring case, punctuation and spaces.

You need to code the palindrome checking logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
public class PalindromeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a character string: ");
        String input = scanner.nextLine();
        boolean isPalindrome = isPalindrome(input);
        if (isPalindrome) {
            System.out.println("The string is a palindrome.");
```

```
        } else {
            System.out.println("The string is not a palindrome.");
        }
    }
    public static boolean isPalindrome(String str) {
        // Remove punctuation, case and spaces
        str = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
        int left = 0;
        int right = str.length() - 1;
        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

In this example, we've used an isPalindrome function to check whether a string is a palindrome. The program asks the user to enter a string, then uses the isPalindrome function to remove punctuation, ignore case and spaces, and then check whether the string is a palindrome using a two pointer technique. The program displays a message indicating whether or not the string is a palindrome.

STATEMENT: COUNTING VOWELS IN A CHARACTER STRING IN JAVA

Write a Java program that counts the number of vowels (letters a, e, i, o, u) in a string. The user enters a string, and the program counts the number of vowels in the string and displays the result.

You need to code the vowel counting logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class VowelCounter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a character string: ");

        String input = scanner.nextLine();

        int vowelCount = countVowels(input);

        System.out.println("The number of vowels in the string is: " +
vowelCount);

    }
```

```
    public static int countVowels(String str) {
        str = str.toLowerCase(); // Convert the string to lower case to ignore case.
        int count = 0;
        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
                count++;
            }
        }
        return count;
    }
}
```

In this example, we have used a countVowels function to count the number of vowels in a string. The program asks the user to enter a string, then uses the countVowels function to convert the string to lower case, then scans the string character by character to count the vowels. The program then displays the number of vowels in the string.

STATEMENT: CALCULATING THE AVERAGE OF AN ARRAY OF NUMBERS IN JAVA

Write a Java program that calculates the average of the numbers in an array. The user must first enter the size of the table, then enter the numbers one by one. The program then calculates the average of these numbers and displays it.

You need to code the averaging logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class AverageCalculator {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the table: ");

        int size = scanner.nextInt();

        double[] numbers = new double[size];

        double sum = 0;

        System.out.println("Enter the elements of the table:");

        for (int i = 0; i < size; i++) {
```

```
            numbers[i] = scanner.nextDouble();
            sum += numbers[i];
        }
        double average = calculateAverage(sum, size);
        System.out.println("The average number is: " + average);
    }
    public static double calculateAverage(double sum, int count) {
        if (count > 0) {
            return sum / count;
        } else {
            return 0.0; // Avoid division by zero if the array is empty
        }
    }
}
```

In this example, we have used an array of numbers and a for loop to allow the user to enter the elements of the array one by one. The sum of the numbers is calculated at the same time. Then the calculateAverage function is used to calculate the average by dividing the sum by the number of elements in the array. If the array is empty, the average is set to 0 to avoid division by zero. The program then displays the average.

STATEMENT : FINDING THE MEDIAN IN AN UNSORTED ARRAY IN JAVA

Write a Java program that finds the median of an unsorted array of integers. The median is the central value of a set of sorted or unsorted data. You need to find the median in an unsorted array.

You need to code the median search logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
import java.util.Arrays;
public class MedianFinder {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the size of the table: ");
      int size = scanner.nextInt();
      int[] numbers = new int[size];
      System.out.println("Enter the elements of the table:");
      for (int i = 0; i < size; i++) {
```

```
            numbers[i] = scanner.nextInt();
        }
        int median = findMedian(numbers);
        System.out.println("The median of the table is: " + median);
    }
    public static int findMedian(int[] array) {
        // Sort the table
        Arrays.sort(array);
        int size = array.length;
        if (size % 2 == 0) {
            // If the table has an even number of elements, the median is the
average of the two central elements.
            int middleIndex1 = size / 2 - 1;
            int middleIndex2 = size / 2;
            return (array[middleIndex1] + array[middleIndex2]) / 2;
        } else {
            // If the table has an odd number of elements, the median is the central
element.
            int middleIndex = size / 2;
            return array[middleIndex];
        }
    }
}
```

In this example, we used the Arrays.sort method to sort the array of integers, which is essential for finding the median. Then we used logic to determine the median based on the parity of the array size. If the array has an even number of elements, the median is the average of the two central elements. If the array has an odd number of elements, the median is the central element. The program then displays the median.

STATEMENT: CHECKING THE UNIQUENESS OF ELEMENTS IN AN ARRAY IN JAVA

Write a Java program that checks whether all the elements of an array are unique, i.e. there are no duplicates in the array. The user must enter the size of the array and the elements of the array, and the program must determine whether or not all the elements are unique.

You need to code the logic for checking the uniqueness of elements in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
import java.util.HashSet;
public class UniqueElementsChecker {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the size of the table: ");
      int size = scanner.nextInt();
      int[] numbers = new int[size];
      System.out.println("Enter the elements of the table:");
```

```
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }
        boolean hasDuplicates = checkForDuplicates(numbers);
        if (hasDuplicates) {
            System.out.println("There are duplicate elements in the array.");
        } else {
            System.out.println("All the elements in the array are unique.");
        }
    }
    public static boolean checkForDuplicates(int[] array) {
        HashSet<Integer> set = new HashSet<Integer>();
        for (int element : array) {
            if (!set.add(element)) {
                return true; // If the element cannot be added to the set, it is a duplicate.
            }
        }
        return false; // No duplicates found.
    }
}
```

In this example, we have used a HashSet to check the uniqueness of the elements in the array. The program asks the user to enter the size of the array and the elements, then uses the checkForDuplicates function to check whether

the array contains any duplicates. If an element cannot be added to the set (because it is already present), the program concludes that there are duplicates. Otherwise, it displays that all the elements are unique.

STATEMENT: MERGING TWO SORTED ARRAYS IN JAVA

Write a Java program that merges two sorted arrays into a single sorted array. The user must enter the size of the two arrays and their elements, and the program must merge the two arrays so that the resulting array is also sorted.

You need to code the logic for merging the two sorted arrays in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
public class SortedArrayMerger {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the first array: ");
        int size1 = scanner.nextInt();
        int[] array1 = new int[size1];
        System.out.println("Enter the elements of the first sorted array:");
        for (int i = 0; i < size1; i++) {
            array1[i] = scanner.nextInt();
```

```
        }
        System.out.print("Enter the size of the second array: ");
        int size2 = scanner.nextInt();
        int[] array2 = new int[size2];
        System.out.println("Enter the elements of the second sorted array:");
        for (int i = 0; i < size2; i++) {
            array2[i] = scanner.nextInt();
        }
        int[] mergedArray = mergeSortedArrays(array1, array2);
        System.out.println("The sorted result table is:");
        for (int num : mergedArray) {
            System.out.print(num + " ");
        }
    }
    public static int[] mergeSortedArrays(int[] array1, int[] array2) {
        int size1 = array1.length;
        int size2 = array2.length;
        int[] result = new int[size1 + size2];
        int i = 0, j = 0, k = 0;
        while (i < size1 && j < size2) {
            if (array1[i] < array2[j]) {
                result[k++] = array1[i++];
            } else {
```

```
            result[k++] = array2[j++];
        }
    }
    while (i < size1) {
        result[k++] = array1[i++];
    }
    while (j < size2) {
        result[k++] = array2[j++];
    }
    return result;
  }
}
```

In this example, we've used a mergeSortedArrays function to merge two sorted arrays into one resulting sorted array. The program asks the user to enter the size and elements of the two sorted arrays, then uses the mergeSortedArrays function to perform the merge. The result is then displayed. The merge is performed by comparing the elements of the two arrays and placing the smaller one in the resulting array, and so on until all the elements have been merged.

STATEMENT: CALCULATING THE CUMULATIVE SUM OF AN ARRAY IN JAVA

Write a Java program that calculates the cumulative sum of an array of integers. The cumulative sum is an array in which each element is the sum of all the previous elements of the original array, including itself. The user must enter the size of the array and its elements, and the program must calculate and display the cumulative sum array.

You need to code the calculation logic for the cumulative sum table in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class CumulativeSumCalculator {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the table: ");

        int size = scanner.nextInt();

        int[] array = new int[size];
```

```
        int[] cumulativeSum = new int[size];
        System.out.println("Enter the elements of the table:");
        for (int i = 0; i < size; i++) {
            array[i] = scanner.nextInt();
        }
        calculateCumulativeSum(array, cumulativeSum);
        System.out.println("The cumulative sum table is:");
        for (int num : cumulativeSum) {
            System.out.print(num + " ");
        }
    }
    public static void calculateCumulativeSum(int[] input, int[] output) {
        int sum = 0;
        for (int i = 0; i < input.length; i++) {
            sum += input[i];
            output[i] = sum;
        }
    }
}
```

In this example, we have used the calculateCumulativeSum function to calculate the cumulative sum table. The program asks the user to enter the size of the array and the elements of the array, then uses the calculateCumulativeSum function to calculate the cumulative sum. The cumulative sum array is displayed at the end, where each element is the sum of all the previous elements, including itself.

STATEMENT: CONVERTING A DECIMAL NUMBER TO BINARY IN JAVA

Write a Java program that converts a decimal number into its binary representation. The user must enter a decimal number, and the program must calculate the binary representation of this number and display it.

You need to code the logic for converting a decimal number to binary in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class DecimalToBinaryConverter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a decimal number: ");

        int decimalNumber = scanner.nextInt();

        String binaryNumber = convertDecimalToBinary(decimalNumber);

        System.out.println("The binary representation is: " + binaryNumber);

    }
```

```
    public static String convertDecimalToBinary(int decimal) {
        if (decimal == 0) {
            return "0";
        }
        StringBuilder binary = new StringBuilder();
        while (decimal > 0) {
            int remainder = decimal % 2;
            binary.insert(0, remainder); // Add the remainder at the beginning of the string
            decimal = decimal / 2;
        }
        return binary.toString();
    }
}
```

In this example, we have used the convertDecimalToBinary function to convert a decimal number into its binary representation. The program asks the user to enter a decimal number and then uses the function to perform the conversion. The conversion logic consists of repeatedly dividing the decimal number by 2 and adding the remainders (0 or 1) at the beginning of the string to obtain the binary representation. The result is then displayed.

STATEMENT : SEARCHING FOR SUBSTRINGS IN A STRING IN JAVA

Write a Java program that searches for all occurrences of a given substring in a string. The user must enter a string and a substring, and the program must find and display all the positions where the substring appears in the string.

You need to code the substring search logic in Java and make sure your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;

public class SubstringSearch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a character string: ");

        String mainString = scanner.nextLine();

        System.out.print("Enter the substring to search for: ");

        String substring = scanner.nextLine();

        findSubstringPositions(mainString, substring);

    }
```

```
    public static void findSubstringPositions(String mainString, String substring) {
        int lastIndex = 0;
        while (lastIndex != -1) {
            lastIndex = mainString.indexOf(substring, lastIndex);
            if (lastIndex != -1) {
                System.out.println("The substring is at position: " + lastIndex);
                lastIndex++;
            }
        }
    }
}
```

In this example, we have used the findSubstringPositions function to search for all occurrences of a substring in a string. The program asks the user to enter the main string and the substring to search for, then uses the function to find and display all the positions where the substring appears. The indexOf method is used to find the positions of the substring in the main string, and a while loop is used to find all occurrences.

STATEMENT: CALCULATING THE INTERSECTION OF TWO SETS IN JAVA

Write a Java program that calculates the intersection of two sets of integers. The user must enter the elements of the two sets, and the program must determine and display the elements that are present in the two sets (i.e. the intersection).

You need to code the logic for calculating the intersection of sets in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.ArrayList;
public class IntersectionOfSets {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the size of the first set: ");
      int size1 = scanner.nextInt();
```

```
        HashSet<Integer> set1 = new HashSet<Integer>();
        System.out.println("Enter the elements of the first set:");
        for (int i = 0; i < size1; i++) {
            set1.add(scanner.nextInt());
        }
        System.out.print("Enter the size of the second set: ");
        int size2 = scanner.nextInt();
        HashSet<Integer> set2 = new HashSet<Integer>();
        System.out.println("Enter the elements of the second set:");
        for (int i = 0; i < size2; i++) {
            set2.add(scanner.nextInt());
        }
        ArrayList<Integer> intersection = findIntersection(set1, set2);
        System.out.println("The intersection of the two sets is: " + intersection);
    }
    public static ArrayList<Integer> findIntersection(HashSet<Integer> set1,
HashSet<Integer> set2) {
        ArrayList<Integer> intersection = new ArrayList<Integer>();
        for (int num : set1) {
            if (set2.contains(num)) {
                intersection.add(num);
            }
        }
        return intersection;
```

```
    }
}
```

In this example, we have used the findIntersection function to calculate the intersection of two sets of integers. The program asks the user to enter the size and elements of the two sets, then uses the function to find the elements that are present in both sets. The intersection elements are stored in an ArrayList and displayed at the end. The contains method is used to check whether an element is present in a set.

STATEMENT: CALCULATING THE UNION OF TWO SETS IN JAVA

Write a Java program that calculates the union of two sets of integers. The user must enter the elements of the two sets, and the program must determine and display the elements that are present in at least one of the sets (i.e. the union).

You need to code the logic for calculating the union of sets in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.ArrayList;
public class UnionOfSets {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the size of the first set: ");
      int size1 = scanner.nextInt();
      HashSet<Integer> set1 = new HashSet<Integer>();
```

```
        System.out.println("Enter the elements of the first set:");
        for (int i = 0; i < size1; i++) {
            set1.add(scanner.nextInt());
        }
        System.out.print("Enter the size of the second set: ");
        int size2 = scanner.nextInt();
        HashSet<Integer> set2 = new HashSet<Integer>();
        System.out.println("Enter the elements of the second set:");
        for (int i = 0; i < size2; i++) {
            set2.add(scanner.nextInt());
        }
        ArrayList<Integer> union = findUnion(set1, set2);
        System.out.println("The union of the two sets is: " + union);
    }
    public static ArrayList<Integer> findUnion(HashSet<Integer> set1,
HashSet<Integer> set2) {
        ArrayList<Integer> union = new ArrayList<Integer>();
        union.addAll(set1);
        union.addAll(set2);
        return union;
    }
}
```

In this example, we have used the findUnion function to calculate the union of two sets of integers. The program asks the user to enter the size and elements

of the two sets, then uses the function to find the elements that are present in at least one of the sets. The union elements are stored in an ArrayList and displayed at the end. Using the addAll method adds all the elements of the sets to the union.

STATEMENT: CALCULATING THE DIFFERENCE BETWEEN TWO SETS IN JAVA

Write a Java program that calculates the difference between two sets of integers. The user must enter the elements of the two sets, and the program must determine and display the elements that are present in the first set but not in the second (i.e. the difference).

You need to code the logic for calculating the difference of sets in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.ArrayList;
public class DifferenceOfSets {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the size of the first set: ");
      int size1 = scanner.nextInt();
      HashSet<Integer> set1 = new HashSet<Integer>();
```

```
        System.out.println("Enter the elements of the first set:");
        for (int i = 0; i < size1; i++) {
            set1.add(scanner.nextInt());
        }
        System.out.print("Enter the size of the second set: ");
        int size2 = scanner.nextInt();
        HashSet<Integer> set2 = new HashSet<Integer>();
        System.out.println("Enter the elements of the second set:");
        for (int i = 0; i < size2; i++) {
            set2.add(scanner.nextInt());
        }
        ArrayList<Integer> difference = findDifference(set1, set2);
        System.out.println("The difference between the two sets is: " +
difference);
    }
    public static ArrayList<Integer> findDifference(HashSet<Integer> set1,
HashSet<Integer> set2) {
        ArrayList<Integer> difference = new ArrayList<Integer>();
        for (int num : set1) {
            if (!set2.contains(num)) {
                difference.add(num);
            }
        }
        return difference;
```

```
    }
}
```

In this example, we've used the findDifference function to calculate the difference between two sets of integers. The program asks the user to enter the size and elements of the two sets, then uses the function to find the elements that are present in the first set but not in the second. The difference elements are stored in an ArrayList and displayed at the end. The contains method is used to check whether an element is present in the second set.

STATEMENT: CALCULATING THE COMPLEMENT OF A SET IN JAVA

Write a Java program that calculates the complement of a set of integers with respect to a given universal set. The user must enter the elements of the universal set, then the elements of the set whose complement is to be calculated, and the program must determine and display the elements that are present in the universal set but not in the given set (i.e. the complement).

You need to code the logic for calculating the complement of sets in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
java
import java.util.Scanner;
import java.util.HashSet;
import java.util.ArrayList;
public class SetComplement {
   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Enter the size of the universal set: ");
        int sizeUniversal = scanner.nextInt();
        HashSet<Integer> universalSet = new HashSet<Integer>();
        System.out.println("Enter the elements of the universal set:");
        for (int i = 0; i < sizeUniversal; i++) {
            universalSet.add(scanner.nextInt());
        }
        System.out.print("Enter the size of the set whose complement you want to calculate: ");
        int sizeSubset = scanner.nextInt();
        HashSet<Integer> subset = new HashSet<Integer>();
        System.out.println("Enter the elements of the set:");
        for (int i = 0; i < sizeSubset; i++) {
            subset.add(scanner.nextInt());
        }
        ArrayList<Integer> complement = findComplement(universalSet, subset);
        System.out.println("The complement of the given set with respect to the universal set is: " + complement);
    }
    public static ArrayList<Integer> findComplement(HashSet<Integer> universalSet, HashSet<Integer> subset) {
        ArrayList<Integer> complement = new ArrayList<Integer>();
        for (int num : universalSet) {
            if (!subset.contains(num)) {
```

```
            complement.add(num);
        }
    }
    return complement;
  }
}
```

In this example, we have used the findComplement function to calculate the complement of a set to a universal set. The program asks the user to enter the size and elements of both the universal set and the set whose complement is to be calculated, then uses the function to find the elements that are present in the universal set but not in the given set. The complement elements are stored in an ArrayList and displayed at the end. The contains method is used to check whether an element is present in the given set.

STATEMENT: TOPOLOGICAL SORTING OF A DIRECTED GRAPH IN JAVA

Write a Java program that performs a topological sort of a directed graph represented as an adjacency list. Topological sorting is a way of ordering the nodes of a graph so that each node precedes all the nodes it reaches by an edge. The program must take the graph as input and display the topological sort order.

You need to code the topological sorting logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
public class TopologicalSort {
    public static void main(String[] args) {
        // Representation of the directed graph as an adjacency list
```

```
        int numNodes = 6;
        ArrayList<ArrayList<Integer>> graph = new ArrayList<>(numNodes);
        for (int i = 0; i < numNodes; i++) {
            graph.add(new ArrayList<Integer>());
        }
        // Add the edges of the graph
        addEdge(graph, 5, 2);
        addEdge(graph, 5, 0);
        addEdge(graph, 4, 0);
        addEdge(graph, 4, 1);
        addEdge(graph, 2, 3);
        addEdge(graph, 3, 1);
        // Perform the topological sort and display the result
        ArrayList<Integer> topologicalOrder = topologicalSort(graph);
        System.out.println("Topological sort order: " + topologicalOrder);
    }
    public static void addEdge(ArrayList<ArrayList<Integer>> graph, int
source, int destination) {
        graph.get(source).add(destination);
    }
    public static ArrayList<Integer>
topologicalSort(ArrayList<ArrayList<Integer>> graph) {
        int numNodes = graph.size();
        int[] inDegree = new int[numNodes];
```

```
        for (int i = 0; i < numNodes; i++) {
            ArrayList<Integer> neighbors = graph.get(i);
            for (int neighbor : neighbors) {
                inDegree[neighbor]++;
            }
        }
        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < numNodes; i++) {
            if (inDegree[i] == 0) {
                queue.add(i);
            }
        }
        ArrayList<Integer> topologicalOrder = new ArrayList<>();
        while (!queue.isEmpty()) {
            int node = queue.poll();
            topologicalOrder.add(node);
            ArrayList<Integer> neighbors = graph.get(node);
            for (int neighbor : neighbors) {
                inDegree[neighbor]--;
                if (inDegree[neighbor] == 0) {
                    queue.add(neighbor);
                }
            }
```

```
        }
        return topologicalOrder;
    }
}
```

In this example, we have represented a directed graph as an adjacency list and performed a topological sort using an algorithm based on the in-degrees of the nodes. The result is displayed as the topological sort order of the graph. You can customise the graph by modifying the edges to see how the topological sort order changes as a result.

STATEMENT: DEEP TRAVERSAL (DFS) OF A GRAPH IN JAVA

Write a Java program that performs a deep traversal (DFS) of an undirected graph represented as an adjacency list. Deep traversal is a technique for exploring all the nodes of a graph by following the edges as far as possible before going back.

You need to code the path logic in depth in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.ArrayList;
import java.util.Stack;
public class DepthFirstSearch {
   public static void main(String[] args) {
      // Representation of the undirected graph as an adjacency list
      int numNodes = 6;
      ArrayList<ArrayList<Integer>> graph = new ArrayList<>(numNodes);
      for (int i = 0; i < numNodes; i++) {
         graph.add(new ArrayList<Integer>());
      }
```

```
        // Add the edges of the graph
        addEdge(graph, 0, 1);
        addEdge(graph, 0, 2);
        addEdge(graph, 1, 3);
        addEdge(graph, 2, 4);
        addEdge(graph, 3, 5);
        // Perform the depth run from the start node 0
        System.out.println("Depth traversal (DFS) from node 0:");
        depthFirstSearch(graph, 0);
    }
    public static void addEdge(ArrayList<ArrayList<Integer>> graph, int
node1, int node2) {
        graph.get(node1).add(node2);
        graph.get(node2).add(node1);
    }
    public static void depthFirstSearch(ArrayList<ArrayList<Integer>> graph,
int startNode) {
        int numNodes = graph.size();
        boolean[] visited = new boolean[numNodes];
        Stack<Integer> stack = new Stack<>();
        stack.push(startNode);
        while (!stack.isEmpty()) {
            int currentNode = stack.pop();
            if (!visited[currentNode]) {
```

```
            System.out.print(currentNode + " ");
            visited[currentNode] = true;
            ArrayList<Integer> neighbors = graph.get(currentNode);
            for (int neighbor : neighbors) {
                if (!visited[neighbor]) {
                    stack.push(neighbor);
                }
            }
        }
    }
}
```

In this example, we have represented an undirected graph as an adjacency list and performed a deep traversal (DFS) starting from a start node. The result is displayed as the order in which the nodes are visited during the traversal. You can customise the graph by modifying the edges or choosing a different starting node to explore different parts of the graph.

STATEMENT: BREADTH-FIRST GRAPH (BFS) IN JAVA

Write a Java program that performs a breadth-first traversal (BFS) of an undirected graph represented as an adjacency list. Breadth-first traversal is a technique for exploring all the nodes in a graph by starting with a starting node and iteratively moving to its nearest neighbours.

You need to code the breadth logic in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
public class BreadthFirstSearch {
   public static void main(String[] args) {
      // Representation of the undirected graph as an adjacency list
      int numNodes = 6;
      ArrayList<ArrayList<Integer>> graph = new ArrayList<>(numNodes);
      for (int i = 0; i < numNodes; i++) {
         graph.add(new ArrayList<Integer>());
```

```
        }
        // Add the edges of the graph
        addEdge(graph, 0, 1);
        addEdge(graph, 0, 2);
        addEdge(graph, 1, 3);
        addEdge(graph, 2, 4);
        addEdge(graph, 3, 5);
        // Perform the width path from the start node 0
        System.out.println("Browse width (BFS) from node 0:");
        breadthFirstSearch(graph, 0);
    }
    public static void addEdge(ArrayList<ArrayList<Integer>> graph, int
node1, int node2) {
        graph.get(node1).add(node2);
        graph.get(node2).add(node1);
    }
    public static void breadthFirstSearch(ArrayList<ArrayList<Integer>>
graph, int startNode) {
        int numNodes = graph.size();
        boolean[] visited = new boolean[numNodes];
        Queue<Integer> queue = new LinkedList<>();
        queue.add(startNode);
        visited[startNode] = true;
        while (!queue.isEmpty()) {
```

```
            int currentNode = queue.poll();
            System.out.print(currentNode + " ");
            ArrayList<Integer> neighbors = graph.get(currentNode);
            for (int neighbor : neighbors) {
                if (!visited[neighbor]) {
                    queue.add(neighbor);
                    visited[neighbor] = true;
                }
            }
        }
    }
}
```

In this example, we have represented an undirected graph as an adjacency list and performed a breadth-first traversal (BFS) from a start node. The result is displayed as the order in which the nodes are visited during the breadth traversal. You can customise the graph by modifying the edges or choosing a different starting node to explore different parts of the graph.

STATEMENT: DIJKSTRA'S SHORTEST PATH ALGORITHM IN JAVA

Write a Java program that implements Dijkstra's algorithm to find the shortest path between a starting node and all other nodes in a weighted directed graph. Dijkstra's algorithm is widely used to solve the shortest path problem.

You need to code the logic of Dijkstra's algorithm in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.PriorityQueue;
public class DijkstraShortestPath {
   public static void main(String[] args) {
      // Representation of the weighted directed graph as an adjacency list
      int numNodes = 6;
      ArrayList<ArrayList<Node>> graph = new ArrayList<>(numNodes);
      for (int i = 0; i < numNodes; i++) {
         graph.add(new ArrayList<Node>());
```

```
        }
        // Add graph edges with weights
        addWeightedEdge(graph, 0, 1, 2);
        addWeightedEdge(graph, 0, 2, 4);
        addWeightedEdge(graph, 1, 2, 1);
        addWeightedEdge(graph, 1, 3, 7);
        addWeightedEdge(graph, 2, 4, 3);
        addWeightedEdge(graph, 3, 4, 2);
        addWeightedEdge(graph, 4, 5, 5);
        // Start node
        int startNode = 0;
        // Perform Dijkstra's algorithm to find the minimum distances
        int[] distances = dijkstra(graph, startNode);
        // Display minimum distances
        System.out.println("Minimum distances from the start node " +
startNode + " :");
        for (int i = 0; i < numNodes; i++) {
            System.out.println("Node " + i + " : " + distances[i]);
        }
    }
    public static void addWeightedEdge(ArrayList<ArrayList<Node>> graph,
int node1, int node2, int weight) {
        graph.get(node1).add(new Node(node2, weight));
    }
```

```
   public static int[] dijkstra(ArrayList<ArrayList<Node>> graph, int
startNode) {
        int numNodes = graph.size();
        int[] distances = new int[numNodes];
        Arrays.fill(distances, Integer.MAX_VALUE);
        distances[startNode] = 0;
        PriorityQueue<Node> priorityQueue = new PriorityQueue<>();
        priorityQueue.add(new Node(startNode, 0));
        while (!priorityQueue.isEmpty()) {
            Node currentNode = priorityQueue.poll();
            int node = currentNode.node;
            int distance = currentNode.distance;
            if (distance > distances[node]) {
                continues;
            }
            for (Node neighbor : graph.get(node)) {
                int newDistance = distances[node] + neighbor.distance;
                if (newDistance < distances[neighbor.node]) {
                    distances[neighbor.node] = newDistance;
                    priorityQueue.add(new Node(neighbor.node, newDistance));
                }
            }
        }
        return distances;
```

```
    }
    static class Node implements Comparable<Node> {
        int node;
        int distance;
        Node(int node, int distance) {
            this.node = node;
            this.distance = distance;
        }
        @Override
        public int compareTo(Node other) {
            return Integer.compare(this.distance, other.distance);
        }
    }
}
```

In this example, we have represented a weighted directed graph as an adjacency list with weights on the edges, then performed Dijkstra's algorithm starting from a start node. The program displays the minimum distances from the start node to all other nodes. You can customise the graph by changing the edges and weights to see how the minimum distances are calculated accordingly.

STATEMENT: BELLMAN-FORD SHORTEST PATH ALGORITHM IN JAVA

Write a Java program that implements the Bellman-Ford algorithm to find the shortest path between a starting node and all other nodes in a weighted directed graph, including detection of negative cycles. The Bellman-Ford algorithm is used to solve the shortest path problem, including graphs with negative weight edges, while detecting negative cycles.

You need to code the logic of the Bellman-Ford algorithm in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.ArrayList;
import java.util.Arrays;
public class BellmanFordShortestPath {
   public static void main(String[] args) {
      // Representation of the weighted directed graph as an adjacency list
      int numNodes = 5;
      ArrayList<ArrayList<Edge>> graph = new ArrayList<>(numNodes);
      for (int i = 0; i < numNodes; i++) {
         graph.add(new ArrayList<Edge>());
```

```
        }
        // Add graph edges with weights
        addWeightedEdge(graph, 0, 1, 6);
        addWeightedEdge(graph, 0, 2, 7);
        addWeightedEdge(graph, 1, 3, 5);
        addWeightedEdge(graph, 2, 1, -2);
        addWeightedEdge(graph, 2, 4, 2);
        addWeightedEdge(graph, 3, 4, 7);
        addWeightedEdge(graph, 4, 3, -3);
        // Start node
        int startNode = 0;
        // Perform the Bellman-Ford algorithm to find the minimum distances
        int[] distances = bellmanFord(graph, startNode);
        // Display minimum distances
        System.out.println("Minimum distances from the start node " +
startNode + " :");
        for (int i = 0; i < numNodes; i++) {
            System.out.println("Node " + i + " : " + distances[i]);
        }
        // Detect negative cycles
        if (hasNegativeCycle(graph, distances)) {
            System.out.println("The graph contains a negative cycle.");
        } else {
```

```
            System.out.println("The graph does not contain any negative
cycles.");
        }
    }
    public static void addWeightedEdge(ArrayList<ArrayList<Edge>> graph,
int node1, int node2, int weight) {
        graph.get(node1).add(new Edge(node2, weight));
    }
    public static int[] bellmanFord(ArrayList<ArrayList<Edge>> graph, int
startNode) {
        int numNodes = graph.size();
        int[] distances = new int[numNodes];
        Arrays.fill(distances, Integer.MAX_VALUE);
        distances[startNode] = 0;
        for (int i = 0; i < numNodes - 1; i++) {
            for (int node = 0; node < numNodes; node++) {
                for (Edge edge : graph.get(node)) {
                    if (distances[node] != Integer.MAX_VALUE &&
distances[node] + edge.weight < distances[edge.node]) {
                        distances[edge.node] = distances[node] + edge.weight;
                    }
                }
            }
        }
        return distances;
```

```
    }
    public static boolean hasNegativeCycle(ArrayList<ArrayList<Edge>>
graph, int[] distances) {
        int numNodes = graph.size();
        for (int node = 0; node < numNodes; node++) {
            for (Edge edge : graph.get(node)) {
                if (distances[node] != Integer.MAX_VALUE && distances[node] +
edge.weight < distances[edge.node]) {
                    return true;
                }
            }
        }
        return false;
    }
    static class Edge {
        int node;
        int weight;
        Edge(int node, int weight) {
            this.node = node;
            this.weight = weight;
        }
    }
}
```

In this example, we have represented a weighted directed graph as an adjacency list with weights on the edges, then performed the Bellman-Ford algorithm from a starting node. The program displays the minimum distances from the start node to all other nodes and detects negative cycles if present. You can customise the graph by changing the edges and weights to see how the minimum distances and negative cycle detection work accordingly.

STATEMENT: KRUSKAL ALGORITHM FOR THE MINIMAL SPANNING TREE IN JAVA

Write a Java program that implements Kruskal's algorithm to find the minimal spanning tree of an undirected weighted graph. The minimal spanning tree is a subgraph that is a tree, i.e. it is cycle-free, and spans all the nodes of the original graph with minimal total weight.

You need to code the logic of Kruskal's algorithm in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.ArrayList;
import java.util.Collections;
public class KruskalMinimumSpanningTree {
  public static void main(String[] args) {
    // Representation of the undirected weighted graph as a list of edges
    ArrayList<Edge> edges = new ArrayList<>();
    edges.add(new Edge(0, 1, 2));
    edges.add(new Edge(0, 2, 3));
    edges.add(new Edge(1, 2, 1));
```

```
        edges.add(new Edge(1, 3, 7));
        edges.add(new Edge(2, 3, 5));
        edges.add(new Edge(2, 4, 2));
        edges.add(new Edge(3, 4, 7));
        // Perform the Kruskal algorithm to find the minimal spanning tree
        ArrayList<Edge> minimumSpanningTree = kruskal(edges);
        // Display the edges of the minimal spanning tree and their total weight
        int totalWeight = 0;
        System.out.println("Edges of the minimal spanning tree:");
        for (Edge edge : minimumSpanningTree) {
            System.out.println("Node " + edge.node1 + " - Node " + edge.node2
+ " : Weight " + edge.weight);
            totalWeight += edge.weight;
        }
        System.out.println("Total weight of the minimal spanning tree: " +
totalWeight);
    }
    public static ArrayList<Edge> kruskal(ArrayList<Edge> edges) {
        // Sort edges by increasing weight
        Collections.sort(edges);
        int numNodes = 5; // The total number of nodes in the graph
        ArrayList<Edge> minimumSpanningTree = new ArrayList<>();
        DisjointSet disjointSet = new DisjointSet(numNodes);
        for (Edge edge : edges) {
```

```
            int root1 = disjointSet.find(edge.node1);
            int root2 = disjointSet.find(edge.node2);
            if (root1 != root2) {
                minimumSpanningTree.add(edge);
                disjointSet.union(root1, root2);
            }
        }
        return minimumSpanningTree;
    }
    static class Edge implements Comparable<Edge> {
        int node1;
        int node2;
        int weight;
        Edge(int node1, int node2, int weight) {
            this.node1 = node1;
            this.node2 = node2;
            this.weight = weight;
        }
        @Override
        public int compareTo(Edge other) {
            return Integer.compare(this.weight, other.weight);
        }
    }
```

```
}
```

In this example, we represented an undirected weighted graph as a list of edges with weights. We then run Kruskal's algorithm to find the minimal spanning tree. The program displays the edges of the minimal spanning tree and their total weights. You can customise the edges of the graph to see how Kruskal's algorithm finds the minimal spanning tree in different graphs.

STATEMENT: PRIM'S ALGORITHM FOR THE MINIMAL SPANNING TREE IN JAVA

Write a Java program that implements Prim's algorithm for finding the minimal spanning tree of an undirected weighted graph. The minimal spanning tree is a subgraph that is a tree, i.e. it is cycle-free, and spans all the nodes of the original graph with minimal total weight.

You need to code the logic of Prim's algorithm in Java and make sure that your program works correctly. You can also add comments to explain your code.

Example of a solution in Java :

```
import java.util.ArrayList;
import java.util.PriorityQueue;
public class PrimMinimumSpanningTree {
   public static void main(String[] args) {
      // Representation of the undirected weighted graph as an adjacency list
      int numNodes = 5;
      ArrayList<ArrayList<Node>> graph = new ArrayList<>(numNodes);
      for (int i = 0; i < numNodes; i++) {
         graph.add(new ArrayList<Node>());
```

```
        }
        // Add graph edges with weights
        addWeightedEdge(graph, 0, 1, 2);
        addWeightedEdge(graph, 0, 2, 3);
        addWeightedEdge(graph, 1, 2, 1);
        addWeightedEdge(graph, 1, 3, 7);
        addWeightedEdge(graph, 2, 3, 5);
        addWeightedEdge(graph, 2, 4, 2);
        addWeightedEdge(graph, 3, 4, 7);
        // Perform Prim's algorithm to find the minimal spanning tree
        ArrayList<Edge> minimumSpanningTree = prim(graph);
        // Display the edges of the minimal spanning tree and their total weight
        int totalWeight = 0;
        System.out.println("Edges of the minimal spanning tree:");
        for (Edge edge : minimumSpanningTree) {
            System.out.println("Node " + edge.node1 + " - Node " + edge.node2
+ " : Weight " + edge.weight);
            totalWeight += edge.weight;
        }
        System.out.println("Total weight of the minimal spanning tree: " +
totalWeight);
    }
    public static void addWeightedEdge(ArrayList<ArrayList<Node>> graph,
int node1, int node2, int weight) {
        graph.get(node1).add(new Node(node2, weight));
```

```
        graph.get(node2).add(new Node(node1, weight));
    }
    public static ArrayList<Edge> prim(ArrayList<ArrayList<Node>> graph)
{
        int numNodes = graph.size();
        ArrayList<Edge> minimumSpanningTree = new ArrayList<>();
        PriorityQueue<Edge> priorityQueue = new PriorityQueue<>();
        boolean[] visited = new boolean[numNodes];
        // Start with any node
        int startNode = 0;
        visited[startNode] = true;
        for (Node neighbor : graph.get(startNode)) {
            priorityQueue.add(new Edge(startNode, neighbor.node,
neighbor.weight));
        }
        while (minimumSpanningTree.size() < numNodes - 1) {
            Edge edge = priorityQueue.poll();
            int node1 = edge.node1;
            int node2 = edge.node2;
            if (visited[node1] && visited[node2]) {
                continues;
            }
            visited[node1] = true;
            visited[node2] = true;
```

```
            minimumSpanningTree.add(edge);
            for (Node neighbor : graph.get(node2)) {
                if (!visited[neighbor.node]) {
                    priorityQueue.add(new Edge(node2, neighbor.node,
neighbor.weight));
                }
            }
        }
        return minimumSpanningTree;
    }
    static class Node {
        int node;
        int weight;
        Node(int node, int weight) {
            this.node = node;
            this.weight = weight;
        }
    }
    static class Edge implements Comparable<Edge> {
        int node1;
        int node2;
        int weight;
        Edge(int node1, int node2, int weight) {
            this.node1 = node1;
```

```
            this.node2 = node2;
            this.weight = weight;
        }
        @Override
        public int compareTo(Edge other) {
            return Integer.compare(this.weight, other.weight);
        }
    }
}
```

In this example, we represented an undirected weighted graph as an adjacency list with weights on the edges. We then ran Prim's algorithm to find the minimal spanning tree. The program displays the edges of the minimal spanning tree and their total weights. You can customise the graph edges to see how Prim's algorithm finds the minimal spanning tree in different graphs.

STATEMENT: TRAVELLING SALESMAN ALGORITHM (TSP) IN JAVA

The Travelling Salesman Algorithm (TSP) is a combinatorial optimisation problem that involves finding the shortest route through a set of cities and back to the starting city. In this exercise, you will implement a solution to solve the TSP using the brute force search algorithm.

PART 1 - STATEMENT :

Write a Java program to solve the TSP problem using the exhaustive search method. You will need to find the optimal permutation of cities (the order in which the traveller visits them) that minimises the total distance travelled. You can represent the distances between cities in the form of a matrix or an adjacency list.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the exhaustive search method to solve the TSP with a matrix of distances between cities:

```
import java.util.ArrayList;
import java.util.List;
public class TSPBruteForce {
  public static void main(String[] args) {
    int[][] distances = {
      {0, 29, 20, 21},
      {29, 0, 15, 17},
      {20, 15, 0, 28},
      {21, 17, 28, 0}
    };
    List<Integer> tour = new ArrayList<>();
    for (int i = 0; i < distances.length; i++) {
      tour.add(i);
    }
    List<Integer> optimalTour = null;
    int minDistance = Integer.MAX_VALUE;
    do {
```

```
            int tourDistance = calculateTourDistance(tour, distances);
            if (tourDistance < minDistance) {
                minDistance = tourDistance;
                optimalTour = new ArrayList<>(tour);
            }
        } while (nextPermutation(tour));
        System.out.println("Optimal tour: " + optimalTour);
        System.out.println("Minimum total distance: " + minDistance);
    }
    public static int calculateTourDistance(List<Integer> tour, int[][]
distances) {
        int distance = 0;
        for (int i = 0; i < tour.size() - 1; i++) {
            distance += distances[tour.get(i)][tour.get(i + 1)];
        }
        distance += distances[tour.get(tour.size() - 1)][tour.get(0)]; // Return to
the starting city
        return distance;
    }
    public static boolean nextPermutation(List<Integer> tour) {
        int n = tour.size();
        int i = n - 2;
        while (i >= 0 && tour.get(i) >= tour.get(i + 1)) {
            i--;
```

```
        }
        if (i < 0) {
            return false;
        }
        int j = n - 1;
        while (tour.get(j) <= tour.get(i)) {
            j--;
        }
        swap(tour, i, j);
        reverse(tour, i + 1, n - 1);
        return true;
    }
    public static void swap(List<Integer> tour, int i, int j) {
        int temp = tour.get(i);
        tour.set(i, tour.get(j));
        tour.set(j, temp);
    }
    public static void reverse(List<Integer> tour, int start, int end) {
        while (start < end) {
            swap(tour, start, end);
            start++;
            end--;
        }
```

```
    }
}
```

This solution uses the exhaustive search method to find the optimal tour for the travelling salesman. It generates all possible permutations of cities, calculates the total distance for each permutation and identifies the permutation with the minimum distance. You can customise the distance matrix to solve other instances of the TSP problem.

STATEMENT : FINDING CYCLES IN A GRAPH IN JAVA

The aim of this exercise is to write a Java program to detect the presence of cycles in a directed or undirected graph. You can choose to work with one of the graph types.

PART 1 - STATEMENT :

Write a Java program that allows a user to enter or represent a graph with its vertices and edges, and then detects whether it contains cycles.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java for detecting cycles in an undirected graph using depth-first search (DFS):

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
public class GraphCycleDetection {
    private int V; // Number of vertices
    private List<List<Integer>> adj; // Adjacency list
    public GraphCycleDetection(int vertices) {
        V = vertices;
        adj = new ArrayList<>(vertices);
        for (int i = 0; i < vertices; i++) {
            adj.add(new ArrayList<>());
        }
    }
    public void addEdge(int u, int v) {
        adj.get(u).add(v);
        adj.get(v).add(u);
```

```
    }
    public boolean hasCycle() {
        Set<Integer> visited = new HashSet<>();
        for (int v = 0; v < V; v++) {
            if (!visited.contains(v)) {
                if (hasCycleUtil(v, -1, visited)) {
                    return true;
                }
            }
        }
        return false;
    }
    private boolean hasCycleUtil(int v, int parent, Set<Integer> visited) {
        visited.add(v);
        for (int neighbor : adj.get(v)) {
            if (!visited.contains(neighbor)) {
                if (hasCycleUtil(neighbor, v, visited)) {
                    return true;
                }
            } else if (neighbor != parent) {
                return true;
            }
        }
```

```
        return false;
    }
    public static void main(String[] args) {
        int V = 4;
        GraphCycleDetection graph = new GraphCycleDetection(V);
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 2);
        graph.addEdge(2, 0);
        graph.addEdge(2, 3);
        graph.addEdge(3, 3);
        if (graph.hasCycle()) {
            System.out.println("The graph contains a cycle.");
        } else {
            System.out.println("The graph contains no cycles.");
        }
    }
}
```

This example shows how to detect cycles in an undirected graph using depth-first search (DFS). The program lets you create a graph, add edges and then check it for cycles. You can customise the graph by adding or modifying edges to test different scenarios.

STATEMENT: BUBBLE SORTING OPTIMISED IN JAVA

The exercise involves writing a Java program that implements the optimised bubble sort algorithm to sort an array of integers in ascending order. Bubble Sort is a simple algorithm that traverses the array, comparing neighbouring elements and swapping them if necessary to order them correctly. The algorithm continues to traverse the array until no swaps are made on a complete pass, indicating that the array is sorted.

PART 1 - STATEMENT :

Write a Java program to sort an array of integers using the optimised bubble sort algorithm. Optimised bubble sorting is an improvement on classic bubble sorting that stops the sorting process as soon as it detects that no exchange is necessary during a complete pass.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the optimised bubble sort algorithm to sort an array of integers:

```
public class BubbleSortOptimized {
    public static void main(String[] args) {
        int[] arr = {64, 34, 25, 12, 22, 11, 90};
        bubbleSort(arr);
        System.out.println("Table sorted:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        boolean swapped;
        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
```

```
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        if (!swapped) {
            // If no inversion has been performed, the table is already sorted.
            break;
        }
    }
  }
}
```

This example solution in Java implements the optimised bubble sort algorithm to sort an array of integers. The program takes an array as input, performs the optimised bubble sort and then displays the sorted array. You can customise the input array to test the optimised bubble sort with different data sets.

STATEMENT: BUCKET SORT IN JAVA

The exercise involves writing a Java program that implements the Bucket Sort algorithm to sort an array of integers in ascending order. Bucket Sort is a non-comparative sorting algorithm that divides the elements of the array into 'drawers' or 'buckets' according to their value, then sorts these drawers individually. It then combines the sorted drawers to obtain the final sorted array.

PART 1 - STATEMENT :

Write a Java program to sort an array of integers using the pigeonhole sorting algorithm. The program must take into account the range of possible values of the integers in the array in order to distribute the elements into the drawers.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the pigeonhole sorting algorithm to sort an array of integers:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class BucketSort {
    public static void main(String[] args) {
        int[] arr = {64, 34, 25, 12, 22, 11, 90, 80, 5};
        bucketSort(arr);
        System.out.println("Table sorted:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void bucketSort(int[] arr) {
        int min = arr[0];
        int max = arr[0];
        int n = arr.length;
```

```
    // Find the minimum and maximum values in the table
    for (int i = 1; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
        } else if (arr[i] > max) {
            max = arr[i];
        }
    }
    int range = max - min + 1;
    // Create buckets
    List<List<Integer>> buckets = new ArrayList<>(range);
    for (int i = 0; i < range; i++) {
        buckets.add(new ArrayList<>());
    }
    // Place items in drawers
    for (int i = 0; i < n; i++) {
        buckets.get(arr[i] - min).add(arr[i]);
    }
    // Sort the items in each drawer
    for (int i = 0; i < range; i++) {
        Collections.sort(buckets.get(i));
    }
    // Combine the sorted elements of the drawers to obtain the sorted array
```

```
        int index = 0;
        for (int i = 0; i < range; i++) {
            for (int num : buckets.get(i)) {
                arr[index++] = num;
            }
        }
    }
}
```

This example solution in Java uses the pigeonhole sorting algorithm to sort an array of integers. The program first determines the range of possible values, creates drawers for each possible value, places the elements in the drawers, sorts the elements in each drawer and then combines the sorted elements to obtain the final sorted array. You can customise the input array to test pigeonhole sorting with different data sets.

STATEMENT: SORTING BY COUNTING IN JAVA

The exercise involves writing a Java program that implements the Counting Sort algorithm to sort an array of integers in ascending order. Counting Sort is a non-comparative sorting algorithm that counts the number of occurrences of each element in the array, then constructs a new sorted array based on these counts.

PART 1 - STATEMENT :

Write a Java program to sort an array of integers using the count sort algorithm. The program needs to determine the range of possible values in the array to determine the number of "counters" required.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the counting sort algorithm to sort an array of integers:

```
public class CountingSort {
    public static void main(String[] args) {
        int[] arr = {4, 2, 2, 8, 3, 3, 1};
        countingSort(arr);
        System.out.println("Table sorted:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void countingSort(int[] arr) {
        int n = arr.length;
        int max = arr[0];
        int min = arr[0];
        // Find the maximum and minimum values in the table
        for (int i = 1; i < n; i++) {
            if (arr[i] > max) {
                max = arr[i];
```

```
        } else if (arr[i] < min) {
            min = arr[i];
        }
    }
    int range = max - min + 1;
    int[] count = new int[range];
    int[] output = new int[n];
    // Count the occurrences of each value
    for (int i = 0; i < n; i++) {
        count[arr[i] - min]++;
    }
    // Update counters to indicate the last position of each value
    for (int i = 1; i < range; i++) {
        count[i] += count[i - 1];
    }
    // Build the sorted table using counters
    for (int i = n - 1; i >= 0; i--) {
        output[count[arr[i] - min] - 1] = arr[i];
        count[arr[i] - min]--;
    }
    // Copy the sorted table to the original table
    System.arraycopy(output, 0, arr, 0, n);
}
```

```
}
```

This example solution in Java uses the count sort algorithm to sort an array of integers. The program first determines the range of possible values, counts the occurrences of each value and then constructs a sorted array using the counters. You can customise the input array to test count sorting with different data sets.

STATEMENT: COMB SORT IN JAVA

The exercise involves writing a Java program that implements the Comb Sort algorithm to sort an array of integers in ascending order. Comb Sort is a variation of Bubble Sort that swaps elements at a fixed distance instead of swapping adjacent elements, making it more efficient.

PART 1 - STATEMENT :

Write a Java program to sort an array of integers using the comb sort algorithm. Comb sorting starts with a large gap between the elements to be compared, then progressively reduces this gap until it reaches zero, thus carrying out comparisons and exchanges efficiently.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the comb sort algorithm to sort an array of integers:

```
public class CombSort {
    public static void main(String[] args) {
        int[] arr = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0};
        combSort(arr);
        System.out.println("Table sorted:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void combSort(int[] arr) {
        int n = arr.length;
        int gap = n;
        double shrink = 1.3; // shrink factor
        boolean swapped = true;
        while (gap > 1 || swapped) {
            // Apply the reduction factor to the gap
            gap = (int) (gap / shrink);
```

```
            if (gap < 1) {
                gap = 1;
            }
            swapped = false;
            for (int i = 0; i + gap < n; i++) {
                if (arr[i] > arr[i + gap]) {
                    int temp = arr[i];
                    arr[i] = arr[i + gap];
                    arr[i + gap] = temp;
                    swapped = true;
                }
            }
        }
    }
}
```

This example solution in Java implements the comb sort algorithm to sort an array of integers. The program starts with a large gap between the elements to be compared, then progressively reduces this gap using a reduction factor. You can customise the input array to test comb sorting with different data sets.

STATEMENT: SORTING BY HEAP (HEAPSORT) IN JAVA

The exercise involves writing a Java program that implements the heapsort algorithm to sort an array of integers in ascending order. Heapsort is a sorting algorithm based on the heap data structure, which allows the array to be sorted efficiently.

PART 1 - STATEMENT :

Write a Java program to sort an array of integers using the heap sort algorithm. The algorithm starts by creating a heap from the array, then extracts the elements from the heap one by one and places them in the sorted array.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the heap sort algorithm to sort an array of integers:

```
public class HeapSort {
    public static void main(String[] args) {
        int[] arr = {8, 4, 1, 56, 3, 44, 23, 6, 28, 0};
        heapSort(arr);
        System.out.println("Table sorted:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void heapSort(int[] arr) {
        int n = arr.length;
        // Create a max heap from the table
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }
        // Extract elements from the heap one by one
        for (int i = n - 1; i > 0; i--) {
```

```
            // Exchange the root (maximum element) with the current element
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            // Call heapify to reduce the size of the unsorted heap
            heapify(arr, i, 0);
        }
    }
    public static void heapify(int[] arr, int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        if (left < n && arr[left] > arr[largest]) {
            largest = left;
        }
        if (right < n && arr[right] > arr[largest]) {
            largest = right;
        }
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
```

```
        }
    }
}
```

This example solution in Java implements the heap sort algorithm to sort an array of integers. The program first creates a max heap from the array, then extracts the elements from the heap one by one and places them in the sorted array. You can customise the input array to test heap sorting with different data sets.

STATEMENT : TERNARY SEARCH IN A SORTED ARRAY IN JAVA

The exercise involves writing a Java program to implement the ternary search algorithm in a sorted array. The ternary search is a variant of the binary search which divides the array into three equal parts and compares the element being searched with the elements in the middle two thirds.

PART 1 - STATEMENT :

Write a Java program to perform a ternary search in a sorted array of integers. The ternary search algorithm divides the array into three equal parts and compares the element searched for with the elements in the middle two thirds. If the element is found, return its index. Otherwise, indicate that the element is not present in the array.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java for a ternary search in a sorted array:

```
public class TernarySearch {
    public static void main(String[] args) {
        int[] sortedArray = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
        int target = 13;
        int result = ternarySearch(sortedArray, target);
        if (result == -1) {
            System.out.println("Element not found in array.");
        } else {
            System.out.println("Element found at index: " + result);
        }
    }
    public static int ternarySearch(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;
        while (left <= right) {
            int partitionSize = (right - left) / 3;
            int mid1 = left + partitionSize;
```

```
            int mid2 = right - partitionSize;
            if (arr[mid1] == target) {
                return mid1;
            } else if (arr[mid2] == target) {
                return mid2;
            } else if (target < arr[mid1]) {
                right = mid1 - 1;
            } else if (target > arr[mid2]) {
                left = mid2 + 1;
            } else {
                left = mid1 + 1;
                right = mid2 - 1;
            }
        }
        return -1; // Element not found
    }
}
```

This example solution in Java implements a ternary search in a sorted array. The program takes a sorted array and a target element as input, then performs the ternary search to find the element. If the element is found, its index is returned. Otherwise, a message indicating that the element was not found is displayed. You can customise the sorted array and target value to perform ternary searches with different data sets.

STATEMENT: EXPONENTIAL SEARCH ALGORITHM IN JAVA

The exercise involves writing a Java program to implement the exponential search algorithm. The exponential search algorithm is used to search for an element in a sorted array by recursively dividing the array into sub-arrays of exponential size.

PART 1 - STATEMENT :

Write a Java program to perform an exponential search in a sorted array of integers. The exponential search algorithm recursively divides the array into sub-arrays of exponential size and compares the element searched for with the element in the middle of each sub-array. If the element is found, return its index. Otherwise, indicate that the element is not present in the array.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java for the exponential search algorithm:

```
public class ExponentialSearch {
    public static void main(String[] args) {
        int[] sortedArray = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
        int target = 13;
        int result = exponentialSearch(sortedArray, target);
        if (result == -1) {
            System.out.println("Element not found in the array.");
        } else {
            System.out.println("Element found at index: " + result);
        }
    }
    public static int exponentialSearch(int[] arr, int target) {
        int n = arr.length;
        // If the element is present in the first position
        if (arr[0] == target) {
            return 0;
        }
```

```
        // Find the range for the exponential search
        int i = 1;
        while (i < n && arr[i] <= target) {
            i *= 2;
        }
        // Apply the binary search to the range found
        return binarySearch(arr, target, i / 2, Math.min(i, n - 1));
    }
    public static int binarySearch(int[] arr, int target, int left, int right) {
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target) {
                return mid;
            }
            if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1; // Element not found
    }
}
```

This example solution in Java implements the exponential search algorithm. The program takes a sorted array and a target element as input, then performs the exponential search to find the element. If the element is found, its index is returned. Otherwise, a message indicating that the element was not found is displayed. You can customise the sorted array and target value to perform exponential searches with different data sets.

STATEMENT: GNOME SORT IN JAVA

The exercise involves writing a Java program that implements the gnome sort algorithm (Gnome Sort) to sort an array of integers in ascending order. Gnome Sort is a simple sort-in-place algorithm that moves the elements of the array from left to right until the array is sorted.

PART 1 - STATEMENT :

Write a Java program to sort an array of integers using the gnome sort algorithm. The gnome sort algorithm starts at the first position in the array, then moves the elements to the right until they are correctly sorted.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the gnome sorting algorithm to sort an array of integers:

```
public class GnomeSort {
    public static void main(String[] args) {
        int[] arr = {8, 4, 1, 56, 3, 44, 23, 6, 28, 0};
        gnomeSort(arr);
        System.out.println("Table sorted:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void gnomeSort(int[] arr) {
        int n = arr.length;
        int index = 0;
        while (index < n) {
            if (index == 0) {
                index++;
            }
            if (arr[index] >= arr[index - 1]) {
```

```
            index++;
        } else {
            // Exchange items to sort them
            int temp = arr[index];
            arr[index] = arr[index - 1];
            arr[index - 1] = temp;
            index--;
        }
    }
  }
}
```

This example solution in Java implements the gnome sort algorithm to sort an array of integers. The program starts at the first position in the array, then moves the elements to the right until they are correctly sorted. You can customise the input array to test gnome sorting with different data sets.

STATEMENT: PANCAKE SORT IN JAVA

The exercise involves writing a Java program that implements the pancake sort algorithm (Pancake Sort) to sort an array of integers in ascending order. Pancake Sort is a sorting algorithm that returns the elements of the array in order to sort them.

PART 1 - STATEMENT :

Write a Java program to sort an array of integers using the pancake sort algorithm. The pancake sort algorithm returns the elements of the array to sort them progressively. You can only return elements from the first position in the array.

PART 2 - EXAMPLE OF A SOLUTION IN JAVA :

Here is an example of a solution in Java that uses the pancake sort algorithm to sort an array of integers:

```
public class PancakeSort {
    public static void main(String[] args) {
        int[] arr = {8, 4, 1, 56, 3, 44, 23, 6, 28, 0};
        pancakeSort(arr);
        System.out.println("Table sorted:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
    public static void pancakeSort(int[] arr) {
        int n = arr.length;
        for (int currentSize = n; currentSize > 1; currentSize--) {
            int maxIndex = findMaxIndex(arr, currentSize);
            if (maxIndex != currentSize - 1) {
                // Invert elements from the beginning to the maximum element
                flip(arr, maxIndex);
```

```
            // Invert all elements of the current array
            flip(arr, currentSize - 1);
        }
    }
}
public static int findMaxIndex(int[] arr, int n) {
    int maxIndex = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] > arr[maxIndex]) {
            maxIndex = i;
        }
    }
    return maxIndex;
}
public static void flip(int[] arr, int end) {
    int start = 0;
    while (start < end) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
```

```
    }
}
```

This example solution in Java implements the pancake sort algorithm to sort an array of integers. The program uses rollover operations to sort the array progressively. You can customise the input array to test pancake sorting with different data sets.

CONCLUSION

Congratulations, you've been on an exciting journey through the world of algorithms using Java as your ally. By working through the 50 exercises in this book, you have gained a solid understanding of the fundamental concepts of programming and problem solving.